Abstract

Code-reuse attacks, such as return-oriented programming (ROP), are a class of buffer overflow attacks that repurpose existing executable code towards malicious purposes. These attacks bypass defenses against code injection attacks by chaining together sequence of instructions, commonly known as gadgets, to execute the desired attack logic. A common feature of these attacks is the reliance on the knowledge of memory layout of the executable code. We propose a fine grained randomization based approach that breaks these assumptions by modifying the layout of the executable code and hinders code-reuse attack. Our solution, Marlin, randomizes the internal structure of the executable code by randomly shuffling the function blocks in the target binary. This denies the attacker the necessary a priori knowledge of instruction addresses for constructing the desired exploit payload. Our approach can be applied to any ELF binary and every execution of this binary uses a different randomization. We have integrated Marlin...
into the bash shell that randomizes the target executable before launching it. Our work shows that such an approach incurs low overhead and significantly increases the level of security against code-reuse based attacks.

**Existing System**

This work extends a preliminary approach that was presented. Specifically, we have made the following extensions. First, the earlier approach was applicable only to limited type of binaries, that is, the binaries that do not use function pointers. Our current work addresses this limitation by handling such binaries as well. Second, the earlier approach provided an offline-tool to perform the randomization. In the current work, we have fully integrated the Marlin technique into a custom bash shell so that the application randomization happens seamlessly when the application is executed using this modified bash shell. We have also implemented a white list approach that allows user to specify the list of binaries that should be randomized. For example, one may wish to randomize only those applications that accept input from a remote user and are thus more vulnerable to
attack. Third, we have improved our earlier implementation of Marlin to make the binary randomization much faster.

**Drawbacks**

- High System Overheads
- Block Shuffling is tough

**Proposed System**

Our proposed solution to defend against code-reuse attacks was to increase the entropy by randomizing the function blocks. One may apply this randomization technique at various levels of granularity: function level, block level or gadget level. The level of granularity to choose is a trade-off between security and performance. In our implementation, we implemented the randomization at the function level which is the most coarse granularity amongst the three mentioned above. However, we show that even this coarse level of granularity provides substantial randomization to make brute force attacks infeasible. Our prototype implementation requires the binary disassembly to contain symbol information, i.e.
a non-stripped binary. In practice however, binaries may be stripped and not contain the symbol information. We address this by using external tools such as Unstrip that restore symbol information to a stripped binary. Another approach to process stripped binaries is to randomize at the level of basic blocks since they do not require function symbols to be identified. However, randomizing at basic block granularity will likely incur higher runtime overhead as it would break the principle of locality.

Advantages

The relative offsets of instructions within the application’s code are constant. That is, if an attacker knows any symbol’s address in the application code, then the location of all gadgets and symbols in application’s codebase is deterministic.

Modules

- Node Creation & Routing
- Code Randomization
- Fixing Jumps and Calls
• Performance Analysis

Module Description

Node Creation & Routing

In this module, a wireless network is created. All the nodes are randomly deployed in the network area. Our network is a mobile network, nodes are assigned with mobility (movement). Source and destination nodes are defined. Data transferred from source node to destination node. Since we are working in mobile network, nodes mobility is set i.e., node move from one position to another.

Code Randomization

Shuffling the function blocks in an application binary changes the relative offsets between instructions that may affect various jump and call instructions. The target destination for these jumps/calls can be specified either as an absolute address or as a relative offset. Relative jumps increment or decrement the program.
counter by a constant value as opposed to absolute jump that directly jump to a fixed address. When the function blocks are randomized, these jumps will no longer point to the desired location and must be ‘fixed’ to point to the proper locations. We achieve this by performing jump patching

Performance Analysis And Result Comparison

For performance evaluation we use the following graph

- Packet delivery ratio
- Throughput
- Delay
IEEE Projects 100% WORKING CODE + DOCUMENTATION+ EXPLANATION – BEST PRICE

LOW PRICE GUARANTEED

System Requirements

Hardware Requirements:-

Processor - Pentium –III

Speed - 1.1 Ghz

RAM - 256 MB(min)

Hard Disk - 20 GB

Key Board - Standard Windows Keyboard

Mouse - Two or Three Button Mouse

Monitor - SVGA
IEEE Projects 100% WORKING CODE + DOCUMENTATION+ EXPLANATION – BEST PRICE

LOW PRICE GUARANTEED

SOFTWARE REQUIREMENTS:-

Operating System : LINUX

Tool : Network Simulator-2

Front End :O TCL (Object Oriented Tool Command Language)

References
